UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA: PROGRAMAÇÃO 2

ALUNO: MARCOS MELO DOS SANTOS

RELATÓRIO DO MILESTONE 2

MACEIÓ/AL, 04 DE MAIO DE 2025

1. Introdução

O projeto *Jackut* é uma rede social baseada no *Orkut* desenvolvida como parte da disciplina de Programação 2. O sistema permite que usuários criem perfis, conectem-se com outros usuários, compartilhem mensagens e gerenciem informações. Esse relatório sintetiza a implementação dos Milestone 1 e 2, composto por 9 *user stories* que são validadas através de testes já construídos e disponibilizados pelo professor.

O sistema foi projetado para oferecer uma experiência robusta e escalável, seguindo princípios de engenharia de software modernos. A escolha cuidadosa de padrões arquiteturais visa garantir a manutenibilidade do código, a facilidade de extensão de funcionalidades e a eficiência nas operações.

2. Objetivos

O desenvolvimento do Jackut teve como meta principal criar uma rede social completa que permita aos usuários estabelecer diferentes tipos de conexões sociais de maneira intuitiva e eficiente. Entre os objetivos específicos destacam-se:

- Prover um sistema seguro para criação e autenticação de contas de usuário, garantindo a privacidade dos dados pessoais.
- Oferecer flexibilidade na customização de perfis, permitindo que os usuários expressem sua identidade digital.
- Implementar um sistema de relacionamentos diversificado, indo além das simples amizades para incluir paqueras, relações fã-ídolo e até mesmo inimizades.
- Facilitar a formação de comunidades temáticas onde os membros possam trocar mensagens e informações relevantes.
- Garantir a persistência confiável de todos os dados do sistema, permitindo que as informações sejam mantidas entre diferentes sessões de uso.

3. Arquitetura do Projeto

A arquitetura do Jackut planejada para separar claramente as responsabilidades em camadas lógicas distintas, promovendo a organização do código e facilitando a manutenção futura.

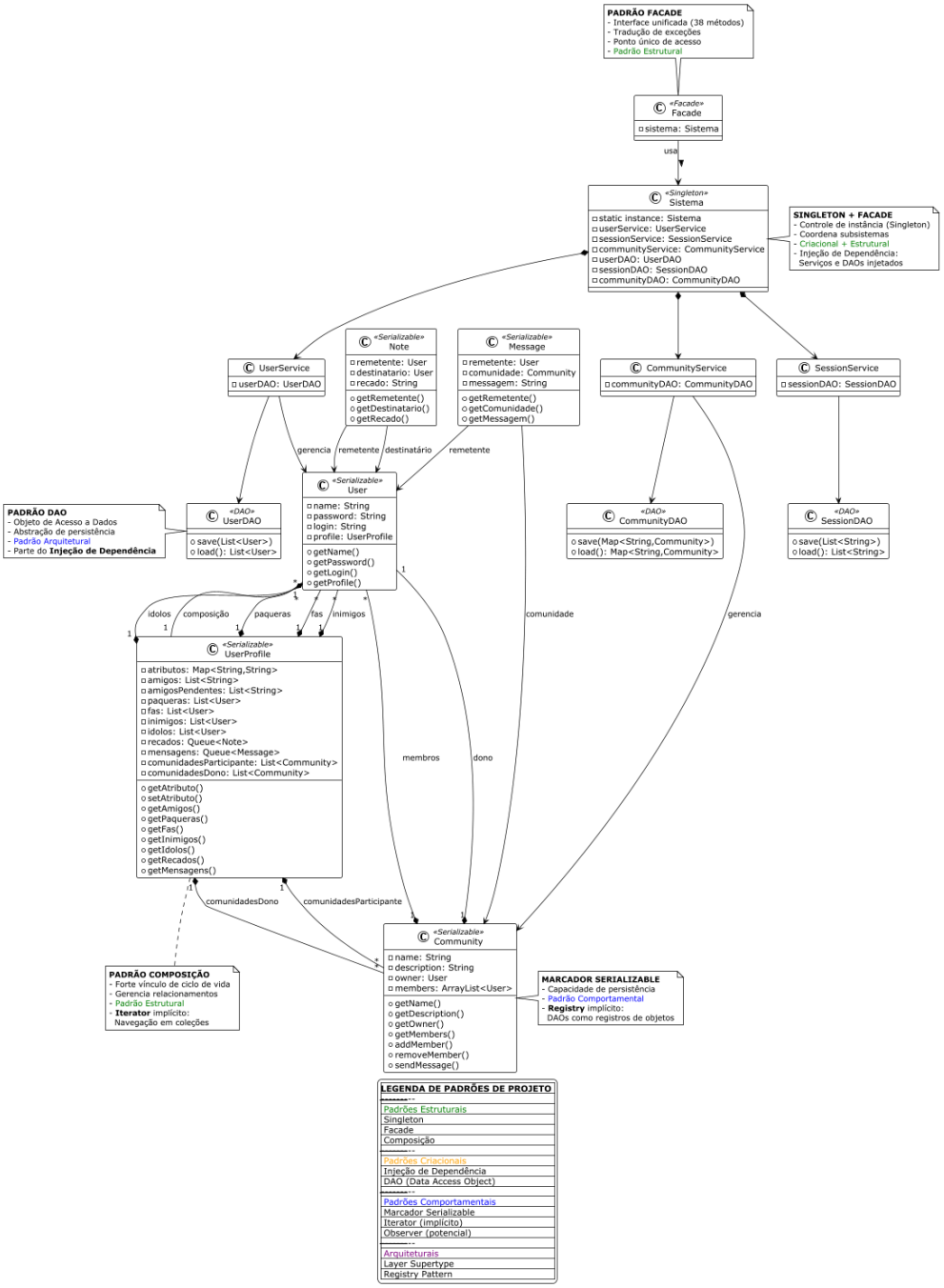
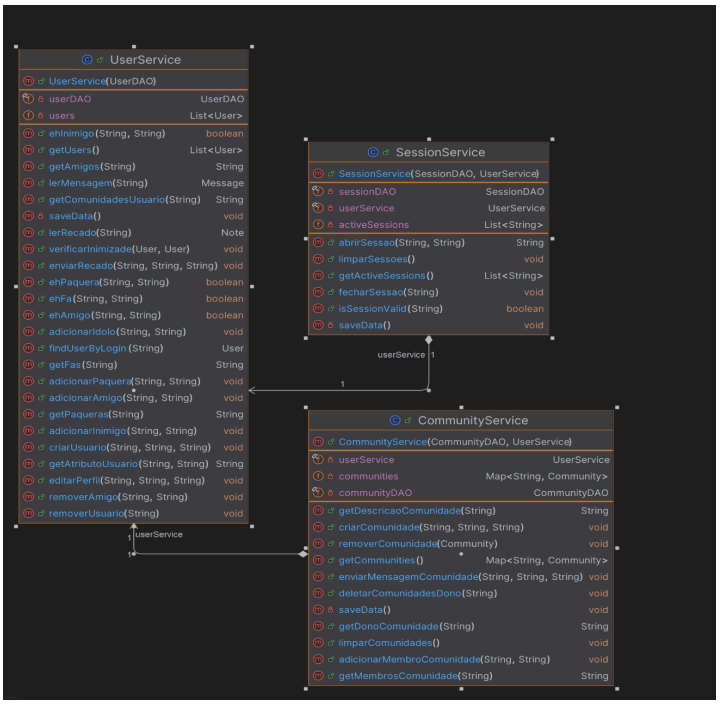


Diagrama de classes

Na camada de domínio encontramos as entidades principais que representam os conceitos fundamentais do sistema. As classes `User` e `UserProfile` encapsulam todas as informações sobre os usuários, incluindo seus perfis e relacionamentos. A classe `Community` modela as comunidades temáticas, enquanto `Message` e `Note` representam respectivamente as mensagens trocadas em comunidades e os recados pessoais entre usuários.

Package Models

A camada de serviço é onde reside a lógica de negócios do sistema. O `UserService` concentra todas as operações relacionadas a usuários, incluindo criação de contas, gerenciamento de perfis e administração de relacionamentos. O `CommunityService` é responsável pelas operações envolvendo comunidades, como criação, adesão de membros e envio de mensagens coletivas. Já o `SessionService` gerencia tudo relacionado a autenticação e sessões de usuário.



Package Services

Para a persistência de dados, o sistema utiliza uma camada `DAO (Data Access Object)` que abstrai as operações de armazenamento. O `UserDAO`, `CommunityDAO` e `SessionDAO` são responsáveis por salvar e recuperar os dados dos respectivos domínios, atualmente utilizando armazenamento em arquivos, mas projetados para permitir fácil migração para outros mecanismos de persistência no futuro.

A camada de apresentação é representada pela classe `Sistema`, que contém a lógica de negócios, serviços e DAOs. Logo após, o `Facade` oferece uma interface unificada e mais simples para interação com as diversas funcionalidades do Jackut.

Classe Sistema

4. Padrões de Design aplicados

A implementação do Jackut incorporou vários padrões de projeto clássicos, cada um resolvendo problemas específicos de design e arquitetura de software.

O padrão `Singleton` foi aplicado à classe Sistema para garantir que exista apenas uma instância do sistema durante a execução da aplicação. Isso é particularmente importante para manter a consistência dos dados e centralizar o controle do estado da aplicação. A implementação inclui verificações de thread safety para uso em ambientes concorrentes.

O padrão `Facade`, como mencionado, é empregado através da classe Sistema que simplifica a interface para o conjunto complexo de subsistemas (serviços de usuário, comunidade e sessão). Isso reduz significativamente a complexidade percebida por outros componentes que precisam interagir com o sistema.

Para o acesso a dados, o padrão `DAO` foi fundamental para separar a lógica de negócios dos detalhes de persistência. Cada entidade principal tem seu DAO correspondente, encapsulando todas as operações de leitura e escrita. Esta abordagem proporciona flexibilidade para alterar o mecanismo de armazenamento sem impactar o restante do sistema.

A `Dependency Injection` é amplamente utilizada na construção dos serviços, permitindo que as dependências (como os DAOs) sejam fornecidas externamente. Isso não apenas facilita os testes unitários, como também torna o sistema mais modular e flexível para extensões futuras.

O padrão `Observer` aparece implicitamente no mecanismo de envio de mensagens para comunidades. Quando uma mensagem é postada em uma comunidade,

todos os membros são automaticamente notificados, demonstrando o padrão Publisher-Subscriber em ação.

Para tratamento de erros, o sistema faz uso de variações do padrão `Factory Method` na criação de exceções específicas. Isso permite padronizar a criação de objetos de exceção enquanto mantém a clareza semântica no código que lida com condições de erro

5. Relações entre funcionalidades e padrões

Cada funcionalidade principal do Jackut foi implementada aproveitando os pontos fortes dos padrões de projeto selecionados.

A criação e autenticação de contas (US 1) utiliza o Singleton para acesso global ao sistema, combinado com o DAO para persistência dos dados do usuário. A injeção de dependência garante que o UserService possa acessar o UserDAO de forma desacoplada.

A edição de perfis (US 2) beneficia-se da arquitetura em camadas, onde a fachada (Sistema) delega as operações para o UserService, que por sua vez coordena as alterações no modelo User e sua persistência via UserDAO.

O sistema de amizades (US 3) implementa um mecanismo de convites que lembra o padrão Observer, onde ações de um usuário geram notificações para outros. A aceitação de um convite de amizade segue um protocolo bem definido que garante a consistência dos relacionamentos.

O envio de recados (US 4) e mensagens para comunidades (Requisito 7) demonstram como o mesmo padrão Observer pode ser aplicado em diferentes contextos - enquanto os recados são direcionados individualmente, as mensagens comunitárias seguem o modelo de broadcast para todos os membros.

A gestão de comunidades (US 5 e 6) ilustra a combinação eficaz dos padrões DAO e Service Layer. O CommunityService orquestra as operações enquanto o CommunityDAO cuida da persistência, mantendo uma separação clara de responsabilidades.

Os relacionamentos alternativos (US 8), como paqueras e relações fã-ídolo, mostram a flexibilidade do sistema em incorporar novos tipos de conexões sociais graças ao design extensível das classes de domínio e serviços.

Por fim, a remoção de contas (US 9) exemplifica como os padrões trabalhando em conjunto garantem a integridade do sistema. O UserService coordena a limpeza completa de todas as referências ao usuário sendo removido, incluindo relacionamentos, mensagens e participação em comunidades, com cada DAO responsável por sua parte específica deste processo complexo.

.

6. Conclusão

O desenvolvimento do Jackut demonstra como a aplicação consciente de padrões de projeto pode resultar em um sistema robusto, bem estruturado e preparado para evoluções futuras. A arquitetura em camadas, combinada com os padrões específicos para cada necessidade funcional, produziu uma base de código que é ao mesmo tempo eficiente no atendimento aos requisitos atuais e flexível para incorporar novas funcionalidades.

Os benefícios desta abordagem são visíveis em vários aspectos. A manutenibilidade é favorecida pela organização lógica dos componentes e pela clara separação de responsabilidades. A testabilidade é ampliada graças ao baixo acoplamento possibilitado pela injeção de dependências. A escalabilidade é assegurada pela arquitetura modular que permite substituir ou estender componentes específicos sem impactar o sistema como um todo.